APPLICATION FOR UNITED STATES LETTERS PATENT

by

MAURICIO LOPEZ

AND

MARK KIRKPATRICK

for a

SYSTEM AND METHOD FOR REAL-TIME APPLICATIONS MODIFICATION

SHAW PITTMAN LLP
1650 Tysons Boulevard
McLean, VA 22102-4859
(703) 770-7900
Attorney Docket No.: BS01-321

SYSTEM AND METHOD FOR REAL-TIME APPLICATIONS MODIFICATION

BACKGROUND

Field of the Invention

[0001] The present invention relates generally to the field of diagnostic tools for computer software. More specifically, the present invention relates to real-time upgrade or other modification of a computer application without terminating or suspending the application's execution.

Background of the Invention

[0002] Web-based software applications often execute on application server platforms. Examples of application server platforms include systems such as iPlanet or Weblogic. Due to their nature, it is desirable to keep web-based applications, highly available. Thus, minimizing their downtime is a critical goal. However, this goal can pose significant problems in the trouble-shooting and maintenance environment of most information technology (IT) facilities. Downtime is often required to debug software problems or to upgrade systems. Downtime is also required to maintain systems or find and repair errors. This downtime can have enormous impacts on IT facilities' operations and viability.

[0003] Conventional software maintenance tools do not adequately solve the problem of modifying or upgrading an application during its normal unimpeded execution. There are several ways of upgrading or modifying applications in conventional systems. In one technique, upgrades or modifications are performed only at scheduled maintenance times. At the scheduled maintenance time, systems personnel are provided a window of time in which any maintenance operations should be

performed. Systems personnel make any required changes, and restart the application within the maintenance window. One problem with the scheduled-maintenance-window approach is that maintenance windows may be scheduled only once or twice a day or even less often. Consequently, there is often a significant delay between the time maintenance is required to resolve non-routine problems, and when the systems personnel are actually allowed to perform it.

[0004] Another problem with the scheduled-maintenance-window approach is that a system operator must be available at the scheduled maintenance time. Due to the dynamic nature of IT facilities, system operators might be assigned to other tasks at the scheduled maintenance time. If a system operator is not available to perform an upgrade during the allotted time, the upgrade must wait until the next scheduled maintenance time. Unfortunately, the next scheduled maintenance time could be hours or days away. Moreover, there is no guarantee that a system operator will be available to complete the required maintenance, because even a simple upgrade may take a significant amount of time to install.

[0005] A second maintenance technique is a reactive strategy – system maintenance or upgrades are performed only when catastrophic failures have occurred or are imminent. Consequently, the second option is used primarily in emergency situations, such as when a system component fails. When the emergency occurs, the application is terminated so that it can be diagnosed or repaired. After the appropriate maintenance or upgrade is completed, the application is restarted using the updated instructions or data.

[0006] The delay between diagnosis and repair of a problem can be significant. Due

to the high premium most web-based applications place on availability, this is an unacceptable solution. Another problem with suspending or terminating an application is that any transaction currently in progress is interrupted, and often lost, when the application is terminated.

[0007] Another conventional technique to handle error conditions is to account for possible errors in the application software in the form of contingency software to handle error conditions. Contingency software enables an application to react to an error in a graceful or predictable manner. The contingency software is embedded in the application. Consequently, the contingency software is compiled as a part of the application and becomes part of the application's executable code.

[0008] One drawback of using the contingency software is that it becomes a fixed part of an application's executable code. Consequently, it is not dynamic. Thus, if an unexpected contingency occurs, there is no software in the program to accommodate it. For this reason, contingency software embedded an application is an incomplete solution.

[0009] Consequently, there is a need for a computer software diagnostic tool that allows system operators to maintain or upgrade a computer application without terminating or suspending the application's operation or adding to the size of executable code.

SUMMARY OF THE INVENTION

[0010] The present invention solves the foregoing problems in the art by providing a system operator with an interface that allows the system operator to upgrade or

otherwise modify a computer application while it is executing. The upgrade or other modification is performed without suspending or terminating the application being analyzed.

[0011] Preferably, an object shell console is used to connect to an executing application. The object shell console extracts program data from the executing application for display to a system operator. For example, the object shell console can obtain and display information regarding classes, methods and fields comprising a particular application. In addition, the value of the variables used in or passed to or from those methods, and any other data that relates to the program structure of the executing application are also displayed to the system operator.

[0012] In one embodiment, the present invention is a system for modifying or upgrading an application during execution without suspending or terminating the application. An application to be analyzed executes on a computer such as an application server. An object shell console executing on an administration client attaches to the application to extract program data from the application without suspending the application or causing the application to terminate. The program data is presented to a system operator in a graphical user interface. A command line is also available to the system operator. The system operator enters a command on the command line. When executed, the command causes modification of the application without suspending or terminating the application.

[0013] In another embodiment, the present invention is a method for modifying or upgrading a computer application while it is executing. The method includes the steps of connecting to the computer application and extracting program data from the

computer application without suspending or terminating the computer application. The method also includes the step of displaying the program data to a system operator in a graphical user interface. The method further includes the step of accepting a command from the system operator. Finally, the method includes the step of executing the command to cause the application to be modified or upgraded without suspending or terminating the application.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] Figure 1 is a schematic diagram a system for modifying a computer application according to an embodiment of the present invention.

[0015] Figure 2 illustrates an exemplary object shell console 601 for modifying an application by re-executing a method with a new parameter according to an embodiment of the present invention.

[0016] Figure 3 illustrates an exemplary object shell console GUI for updating an application to input updated data according to an embodiment of the present invention.

[0017] Figure 4 illustrates an exemplary object shell console GUI for modifying the behavior of an application according to an embodiment of the present invention.

[0018] Figure 5 illustrates another exemplary object shell console GUI for modifying the behavior of an application according to an embodiment of the present application.

[0019] Figure 6 is a flow diagram illustrating a method for modifying a computer application according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0020] Figure 1 is a schematic diagram of a system for allowing system operators to upgrade or maintain a computer application while it is executing, according to an embodiment of the present invention. As used herein, the term "system operator" refers to system administrators, operators, software maintenance personnel, programmers or other persons responsible for, or having authority for, maintaining, upgrading, repairing or otherwise modifying applications. In addition, as used herein, the term "modifying" includes upgrading, repairing, maintaining or otherwise modifying a computer software application. An object shell console 102 executes on an administration client 104. Administration client 104 can be any computer that can execute an object shell console 102 having the functionality described herein, including application server 106. Object shell console 102 attaches to an application executing on an application server 106 so that the application can be modified. Importantly, the connection does not suspend or terminate the application to be modified.

[0021] Exemplary applications 110, 112, 114 and 116 are illustrated executing on an application server 106. One application, for example, is an order application 110. Order application 110 is used to process orders from customers. Order application 110 is coupled to an order database 118. Order database 118 contains order information required to support order application 110. For example, in an embodiment of the present invention, order database 118 contains inventory data, supplier data and customer data.

[0022] In an embodiment of the present invention, object shell console 102 uses

attributes of an interpretive programming language, such as the Java programming language, to connect to an application that is to be modified without terminating or suspending the application. Once connected, object shell console 102 can extract program data from the application. Program data includes any program information related to the application, including, for example, methods, classes, fields, variable names and values, and arguments passed to or from methods. The program data is displayed to a system operator who uses the program data to modify the executing application without terminating or suspending the executing application. In one embodiment of the present invention, object shell console 102 provides this capability by connecting to the Java virtual machine (JVM) on which the application to be modified is executing.

[0023] In one embodiment of the present invention, the connection is made using Java remote method invocation (RMI). Java RMI is a well-known tool that can be used to access one JVM from another JVM. Use of Java RMI enables remote invocation and execution of applications and methods. To invoke a remote application (or method), Java RMI creates a thread to the other application or method. Creation of the new thread using Java RMI occurs without suspending or terminating the executing application.

[0024] The application that is connected to and invoked can be either local or remote to administration client 104. Once the connection is established, object shell console 102 has access to the program data of the invoked application. This access results from a feature of the Java programming language known as introspection. Introspection was developed for Java Beans to allow integrated development

environments (IDEs) to visually manipulate graphical components to build applications. The object language components that are used for introspection are Class, Method and Field. When object shell console 102 retains a reference to a running object by creating the thread to the application, these components are used to extract the fields and execute the methods of the object class. The fields of the object can be manipulated to create different behaviors in the object. Methods can be re-executed with argument values supplied by a system operator. The values returned by the method's execution can be displayed for analysis.

[0025] Object shell console 102 gains this access without terminating or suspending the invoked application. Thus, once connected, object shell console 102 can determine any classes and methods that are present in the attached application without terminating or suspending the attached application. Because object shell console 102 also has access to the methods in an object, those methods can be re-executed by the system operator to affect the behavior of the application without requiring termination or suspension of the application. The effect on the computer on which the application is running caused by the execution of the method is reflected in the application's execution. Thus, if memory is affected, that effect will carry over to the executing application. For example, using the present invention, an operator can change the value of a variable stored in a computer's memory by a method that sets the value of that variable by re-executing. After the method is re-executed, the application will use the new value created by the method.

[0026] As an example, assume that order database 118 crashes and is replaced with new order database 120. New order database 120 is identified by the name "NEW

DB." When the database is replaced, the name must be changed to reflect the change. If the database name is not changed, the application will try to perform database operations on the old database name. As a result, in most cases the application will fail to work properly.

[0027] In some conventional systems, this change requires suspending or terminating the application and editing a file to make the required name change. After the file is edited, the entire application is recompiled and built. Then the application is restarted. At this point, the database name change is complete, and the application begins to function properly.

[0028] In other conventional systems, the change can be performed only at a scheduled maintenance time. At the scheduled maintenance time, a maintenance window is provided that allows a system operator to update the application. In many cases, the application must then be recompiled. Waiting for scheduled maintenance times, or for the application to recompile, can result in significant losses of traffic and revenue for a website.

[0029] Using the present invention, a system operator can modify an application without terminating or suspending the application, and without waiting for a scheduled maintenance time. Moreover, the update occurs substantially in real-time because the re-execution of a method in an interpretive environment such as Java repopulates computer memory with the desired contents.

[0030] An important feature of the present invention is that only the affected method needs to be re-executed. This avoids complex and time-consuming compiles and/or rebuilds of applications. This is a distinct advantage over conventional

systems that generally require an entire application to be rebuilt by recompiling and relinking the method. This rebuilding can be very time-consuming, especially for larger applications. Consequently, only re-executing the affected method as opposed to the entire application generally results in significant time savings that translate into significant reductions in any downtime that might be required.

[0031] An exemplary use of the present invention is described with reference to Figure 2. Figure 2 illustrates an exemplary object shell console GUI application by re-executing a method with a new parameter according to an embodiment of the present invention. As shown in Figure 2, object shell console 102 lists program data associated with a database class. The program data is preferably listed in a program data portion 202 of the GUI of object shell console 102.

[0032] The exemplary database class shown in Figure 2 comprises a READ_ORD() method that allows a user to read an order entry from order database 118, a WRITE_ORD() method that allows a user to write and order entry to order database 118, a DELETE_ORD() method that allows a user to delete an order entry from order database 118 and an UPDATE_ORD() method that allows a user to modify an order entry in order database 118. In addition, the database class comprises two INIT() methods. Each INIT() method has a different number of arguments. It would be apparent to those skilled in the art that Java allows such overloading of methods to accommodate invocation of methods with different numbers of arguments.

[0033] When the methods are listed, object shell console 102 also lists any arguments that were passed when the methods were involved. As a result, a system operator would see the wrong database name in the argument list shown. For example, in the

present example, the program data listing is INIT ("OLD DB"), which indicates that the database was initialized with the identifier "OLD DB."

[0034] Once the system operator discovers the incorrect database name, the system operator can re-execute a method with new parameters to change the value of a variable to fix a problem. To re-execute a method, the system operator enters the method name and correct parameters at a command prompt in a command portion 204 of the GUI of object shell console 102. For example, the system operator can enter DB INIT ("NEWDB") at the command prompt to update the name of the database to reflect the database change.

[0035] Alternatively, the system operator can type the method name with no parameters at the prompt in command portion 204. The system operator is then prompted to enter the parameter(s) associated with the method corresponding to the entered method name. The system operator responds to the prompts by typing any required parameters. When all parameters have been entered, the system operator provides an indication that there are no more parameters to be entered in a well-known manner.

[0036] The method is then re-executed. By invoking the method while object shell console 102 is connected to the application using Java RMI, the memory of the computer on which the application is executing is changed to reflect the new value of the name of the database that was passed in the INIT() method. This update occurs without suspending or terminating the executing application. Because the contents of the memory that the application is using have changed, the executing application continues its execution, using the updated contents of memory. Consequently, the

application is modified substantially in real-time without suspension or termination. By modifying the application in this manner, downtime is minimized, thus furthering a primary goal of web-based applications.

[0037] Figure 3 illustrates an object shell console GUI for updating an application to input updated data according to an embodiment of the present invention. In the system associated with the GUI shown in Figure 3, the contents of database 118 are cached into memory for the system to operate. The cached data comprises the price of each item of inventory in the order database. If the database is changed to update pricing, those changes must be transferred to the data cached in computer memory to affect the application. In conventional systems, the entire application must be suspended and restarted. Restarting the application puts the new price data into the data cache computer memory.

[0038] In another example, of the present invention, the price data of the order system is stored in a data structure that is an object variable. This can be a class variable or other static storage technique. Using the present invention, a system operator can avoid suspending and restarting the application. This is done by first attaching to the application using object shell console 102. Then, the system operator issues a SQL SELECT command in the command line of the GUI of object shell console 102. An exemplary command is SELECT ORDER PRICE. This command extracts the new price data from order database 118. That price information is stored in the static price data structure. The updated data will then be used by any other class components that reference this data structure. Therefore, execution of this method by the object shell console provides new pricing information to all the

application software that accesses this class for price information. This is done without suspending or terminating the application.

[0039] The present invention can also be used to change the behavior of applications without terminating or suspending the application's execution. Figure 4 illustrates an exemplary object shell console GUI for modifying the behavior of an application according to an embodiment of the present invention. In this example, it is desired to generate detailed status messages when errors are detected. If no errors are detected, then short messages only are sufficient.

[0040] While this functionality can be pre-coded into applications, as described above, pre-coding this kind of extra functionality increases the size of the executable that runs in memory as well as increases the size and complexity of diagnostic tools that are used to maintain the system. Using the present invention, however, a programmer does not have to pre-code the extra functionality in the application. Consequently, the executable will be significantly smaller. Moreover, any diagnostic operations, such as debugging, will be greatly simplified due to the reduced size of the executable. Any extra code is stored on a disk. As is well-known to those skilled in the art disk space is relatively inexpensive and plentiful.

[0041] Initially, no errors are expected. So the logging class is invoked with the terse logging. Later an error occurs. When the error occurs, an operator or system administrator invokes the detailed logging as shown in Figure 4. As described above, this is accomplished via the command line. In this case, the command is DB.LOG (DETAIL). After the command is executed, the application will provide detailed logging. This is performed without terminating or suspending the application, or

waiting for a scheduled maintenance time.

[0042] In another embodiment of the present invention, an application's behavior is modified by changing the order of execution of methods in the application without suspending or terminating the application. For example, if an application is written as a series of method calls, the order of execution of these methods can be modified using the present invention. In one embodiment of the present invention, the order of execution of subroutines is passed to the application as a vector. In this case, the vector is an array of numbers containing the desired order of execution.

[0043] As shown in Figure 5, for example, an EXEC_ORD database stores the order of execution of methods M1, M2 and M3. It is assumed that an application is written that executes methods M1, M2 and M3 in the order given in the EXEC_ORD vector. To modify the behavior of the application, the vector EXEC_ORD is modified. For example, in Figure 5, the value of EXEC_ORD is changed from [1,2,3] to [2,1,3]. This means that the order of execution of methods M1, M2, M3 changes from M1, M2, M3 to M2, M1, M3.

[0044] Figure 6 is a flow diagram for a method for upgrading or otherwise modifying a computer application according to an embodiment of the present invention. In step 602, object shell console 102 attaches to an executing application. As described above, one way to accomplish this attachment is by using Java RMI to create a thread that executes the application. In step 604, object console 102 extracts program data from the executing application. In an embodiment of the present invention, step 604 relies on introspection as described above. This program data includes classes, methods, fields and other data comprising the internal program structure of the

application. The program data is displayed to the maintenance person in step 606. Preferably, the program data is displayed in data portion 202 of the GUI of object shell console 102.

[0045] In step 608, object shell console accepts a command from the software maintenance person. Preferably, the command will upgrade or otherwise modify the executing application. Examples of such modifications include affecting data that is processed by the application as well as the behavior of the application. In step 610, object shell console executes the command to modify the application.

[0046] The foregoing disclosure of the preferred embodiments of the present invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many variations and modifications of the embodiments described herein will be apparent to one of ordinary skill in the art in light of the above disclosure. The scope of the invention is to be defined only by the claims appended hereto, and by their equivalents.

[0047] Further, in describing representative embodiments of the present invention, the specification may have presented the method and/or process of the present invention as a particular sequence of steps. However, to the extent that the method or process does not rely on the particular order of steps set forth herein, the method or process should not be limited to the particular sequence of steps described. As one of ordinary skill in the art would appreciate, other sequences of steps may be possible. Therefore, the particular order of the steps set forth in the specification should not be construed as limitations on the claims. In addition, the claims directed to the method

and/or process of the present invention should not be limited to the performance of their steps in the order written, and one skilled in the art can readily appreciate that the sequences may be varied and still remain within the spirit and scope of the present invention.